```python
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
pip install mlxtend seaborn
```

```
Requirement already satisfied: mlxtend in
/usr/local/lib/python3.10/dist-packages (0.22.0)
Requirement already satisfied: seaborn in
/usr/local/lib/python3.10/dist-packages (0.13.1)
Requirement already satisfied: scipy>=1.2.1 in
/usr/local/lib/python3.10/dist-packages (from mlxtend) (1.11.4)
Requirement already satisfied: numpy>=1.16.2 in
/usr/local/lib/python3.10/dist-packages (from mlxtend) (1.25.2)
Requirement already satisfied: pandas>=0.24.2 in
/usr/local/lib/python3.10/dist-packages (from mlxtend) (2.0.3)
Requirement already satisfied: scikit-learn>=1.0.2 in
/usr/local/lib/python3.10/dist-packages (from mlxtend) (1.2.2)
Requirement already satisfied: matplotlib>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from mlxtend) (3.7.1)
Requirement already satisfied: joblib>=0.13.2 in
/usr/local/lib/python3.10/dist-packages (from mlxtend) (1.3.2)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from mlxtend) (67.7.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0-
>mlxtend) (1.2.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0-
>mlxtend) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0-
>mlxtend) (4.50.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0-
>mlxtend) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0-
>mlxtend) (24.0)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0-
>mlxtend) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0-
>mlxtend) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0-
>mlxtend) (2.8.2)
```

```
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->mlxtend)
(2023.4)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->mlxtend)
(2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.2-
>mlxtend) (3.4.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib>=3.0.0->mlxtend) (1.16.0)
```

1. Association Rule Generation from Transaction Data

Part (c) after Part (a) and Part (b)

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
file_path = r"/content/drive/MyDrive/Rowan University 23-25/Spring
'24/Data Mining I/Grocery_Items_24.csv"
df = pd.read_csv(file_path)
transactions = []
for _, row in df.iterrows():
    transactions.append([item for item in row if pd.notna(item)])
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)
min_support = 0.01
frequent_itemsets = apriori(df_encoded, min_support=min_support,
use_colnames=True)
min_confidence = 0.1
rules = association_rules(frequent_itemsets, metric='confidence',
min_threshold=min_confidence)
print("\nAssociation Rules:")
print(rules)


Association Rules:
          antecedents         consequents  antecedent support  \
0        (rolls/buns)  (other vegetables)            0.111875
1        (whole milk)  (other vegetables)            0.156000
2  (other vegetables)        (whole milk)            0.122750
3        (rolls/buns)        (whole milk)            0.111875
4              (soda)        (whole milk)            0.093625

   consequent support   support  confidence      lift  leverage
conviction  \
0             0.12275  0.011250    0.100559  0.819215 -0.002483
```

```
0.975328
1               0.12275  0.016125    0.103365  0.842081 -0.003024
0.978381
2               0.15600  0.016125    0.131365  0.842081 -0.003024
0.971639
3               0.15600  0.013500    0.120670  0.773528 -0.003952
0.959822
4               0.15600  0.011375    0.121495  0.778816 -0.003231
0.960723

   zhangs_metric
0      -0.199025
1      -0.181802
2      -0.176125
3      -0.247927
4      -0.238580
```

Part (d)

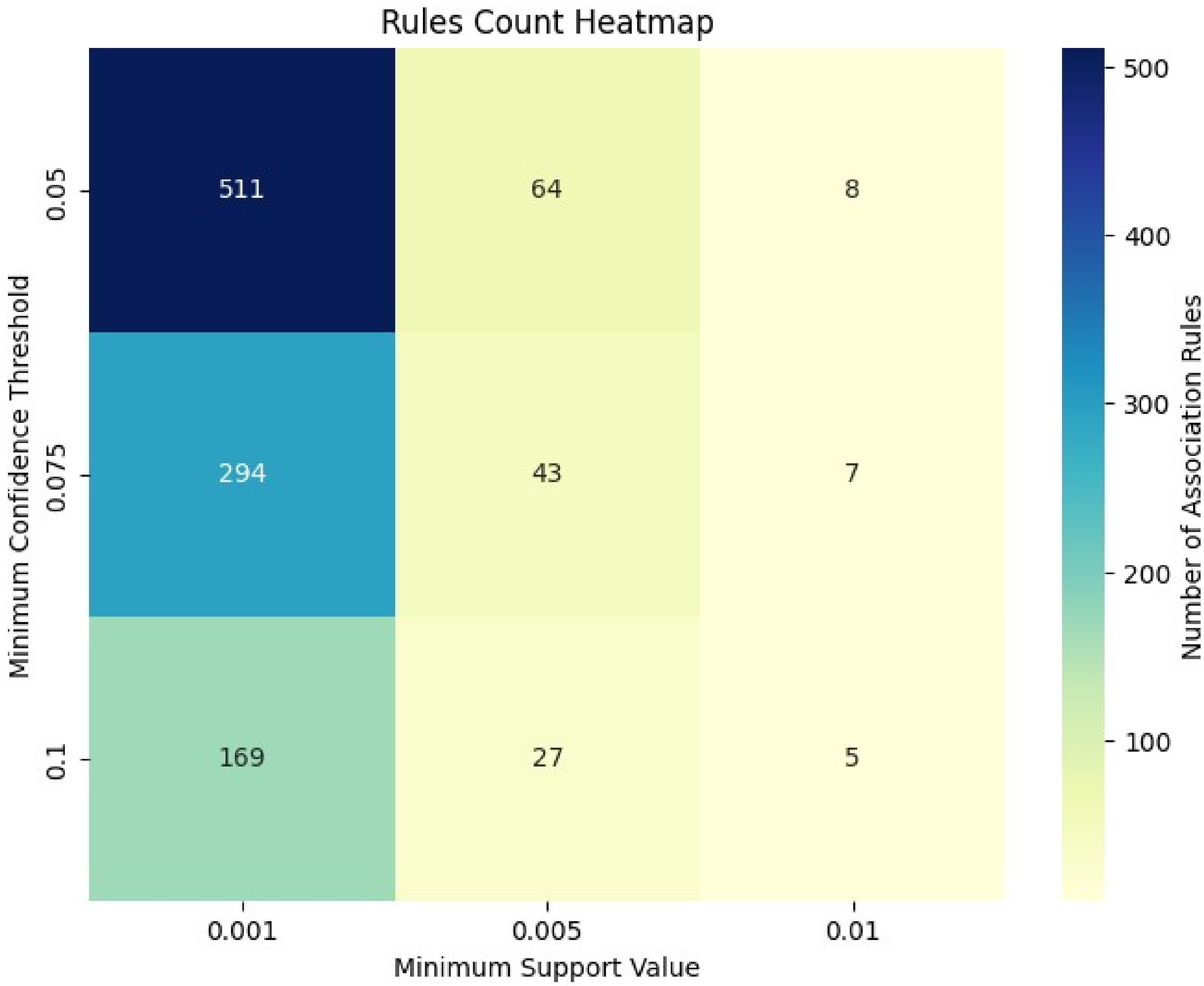
```
import seaborn as sns
import matplotlib.pyplot as plt
from mlxtend.frequent_patterns import apriori, association_rules
import pandas as pd

min_support_values = [0.001, 0.005, 0.01]
min_confidence_thresholds = [0.05, 0.075, 0.1]

count_results = []
for minimum_support in min_support_values:
    for minimum_confidence in min_confidence_thresholds:

        frequent_itemsets = apriori(df_encoded,
min_support=minimum_support, use_colnames=True)

        rules = association_rules(frequent_itemsets,
metric="confidence", min_threshold=minimum_confidence)

        num_rules = len(rules)

        count_results.append({
            'Minimum Support Value': minimum_support,
            'Minimum Confidence Threshold': minimum_confidence,
            'Count': num_rules
        })


        print(f"Minimum Support Value: {minimum_support}, Minimum
Confidence Threshold: {minimum_confidence}")
        print(f"Number of Association Rules: {num_rules}")
        print("===")
```

```python
count_df = pd.DataFrame(count_results)


heatmap_data = count_df.pivot_table(
    index="Minimum Confidence Threshold",
    columns="Minimum Support Value",
    values="Count"
)

plt.figure(figsize=(8, 6))
sns.heatmap(heatmap_data, annot=True, fmt="d", cmap="YlGnBu",
cbar_kws={'label': 'Number of Association Rules'})
plt.title("Rules Count Heatmap")
plt.xlabel("Minimum Support Value")
plt.ylabel("Minimum Confidence Threshold")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

```
Minimum Support Value: 0.001, Minimum Confidence Threshold: 0.05
Number of Association Rules: 511
===
Minimum Support Value: 0.001, Minimum Confidence Threshold: 0.075
Number of Association Rules: 294
===
Minimum Support Value: 0.001, Minimum Confidence Threshold: 0.1
Number of Association Rules: 169
===
Minimum Support Value: 0.005, Minimum Confidence Threshold: 0.05
Number of Association Rules: 64
===
Minimum Support Value: 0.005, Minimum Confidence Threshold: 0.075
Number of Association Rules: 43
===
Minimum Support Value: 0.005, Minimum Confidence Threshold: 0.1
Number of Association Rules: 27
===
Minimum Support Value: 0.01, Minimum Confidence Threshold: 0.05
Number of Association Rules: 8
===
Minimum Support Value: 0.01, Minimum Confidence Threshold: 0.075
Number of Association Rules: 7
===
```

```
Minimum Support Value: 0.01, Minimum Confidence Threshold: 0.1
Number of Association Rules: 5
===
```



Part (e)

```
from sklearn.model_selection import train_test_split


df_subset1, df_subset2 = train_test_split(df_encoded, test_size=0.5,
random_state=42)


min_support_subset = 0.005
min_confidence_subset = 0.075

def extract_association_rules(subset):
    frequent_itemsets = apriori(subset,
min_support=min_support_subset, use_colnames=True)
```

```
    rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=min_confidence_subset)
    return rules

rules_subset1 = extract_association_rules(df_subset1)
print("Association Rules for Subset 1:")
print(rules_subset1)

rules_subset2 = extract_association_rules(df_subset2)
print("\nAssociation Rules for Subset 2:")
print(rules_subset2)

common_rules =
set(rules_subset1.index).intersection(set(rules_subset2.index))

print("\nCommon Association Rules:")
if common_rules:
    for rule_index in common_rules:
        print(rule_index)
else:
    print("No common association rules between the subsets.")
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)

Association Rules for Subset 1:
           antecedents         consequents  antecedent support  \
0       (bottled beer)        (whole milk)             0.04975
1       (bottled beer)            (yogurt)             0.04975
2      (bottled water)  (other vegetables)             0.05975
3      (bottled water)        (whole milk)             0.05975
4        (canned beer)        (whole milk)             0.04650
5       (citrus fruit)  (other vegetables)             0.05675
6       (citrus fruit)        (whole milk)             0.05675
7      (domestic eggs)        (whole milk)             0.03575
8        (frankfurter)  (other vegetables)             0.03900
9          (pip fruit)  (other vegetables)             0.05275
10        (rolls/buns)  (other vegetables)             0.11225
11  (other vegetables)        (rolls/buns)             0.11800
12           (sausage)  (other vegetables)             0.05950
13              (soda)  (other vegetables)             0.08975
14    (tropical fruit)  (other vegetables)             0.06525
15        (whole milk)  (other vegetables)             0.15400
16  (other vegetables)        (whole milk)             0.11800
17            (yogurt)  (other vegetables)             0.08975
18            (pastry)        (whole milk)             0.05500
```

```
19          (pip fruit)        (rolls/buns)               0.05275
20          (pip fruit)        (whole milk)               0.05275
21          (pip fruit)            (yogurt)               0.05275
22            (sausage)        (rolls/buns)               0.05950
23       (shopping bags)       (rolls/buns)               0.04700
24               (soda)        (rolls/buns)               0.08975
25     (tropical fruit)        (rolls/buns)               0.06525
26         (rolls/buns)        (whole milk)               0.11225
27         (whole milk)        (rolls/buns)               0.15400
28         (rolls/buns)            (yogurt)               0.11225
29             (yogurt)        (rolls/buns)               0.08975
30   (root vegetables)         (whole milk)               0.05925
31            (sausage)              (soda)               0.05950
32            (sausage)        (whole milk)               0.05950
33            (sausage)            (yogurt)               0.05950
34       (shopping bags)       (whole milk)               0.04700
35     (tropical fruit)              (soda)               0.06525
36               (soda)        (whole milk)               0.08975
37     (tropical fruit)        (whole milk)               0.06525
38     (tropical fruit)            (yogurt)               0.06525
39             (yogurt)        (whole milk)               0.08975

    consequent support  support  confidence      lift  leverage
conviction  \
0              0.15400  0.00800    0.160804  1.044182  0.000339
1.008108
1              0.08975  0.00500    0.100503  1.119805  0.000535
1.011954
2              0.11800  0.00575    0.096234  0.815545 -0.001300
0.975917
3              0.15400  0.00825    0.138075  0.896593 -0.000951
0.981524
4              0.15400  0.00650    0.139785  0.907694 -0.000661
0.983475
5              0.11800  0.00500    0.088106  0.746659 -0.001697
0.967217
6              0.15400  0.00600    0.105727  0.686538 -0.002740
0.946020
7              0.15400  0.00600    0.167832  1.089819  0.000495
1.016622
8              0.11800  0.00575    0.147436  1.249457  0.001148
1.034526
9              0.11800  0.00525    0.099526  0.843441 -0.000974
0.979484
10             0.11800  0.00950    0.084633  0.717225 -0.003746
0.963547
11             0.11225  0.00950    0.080508  0.717225 -0.003746
0.965479
12             0.11800  0.00625    0.105042  0.890187 -0.000771
```

```
0.985521
13             0.11800  0.00800    0.089136  0.755394 -0.002590
0.968312
14             0.11800  0.00575    0.088123  0.746802 -0.001949
0.967235
15             0.11800  0.01475    0.095779  0.811688 -0.003422
0.975425
16             0.15400  0.01475    0.125000  0.811688 -0.003422
0.966857
17             0.11800  0.00850    0.094708  0.802606 -0.002090
0.974271
18             0.15400  0.00525    0.095455  0.619835 -0.003220
0.935276
19             0.11225  0.00575    0.109005  0.971089 -0.000171
0.996358
20             0.15400  0.00650    0.123223  0.800148 -0.001623
0.964897
21             0.08975  0.00500    0.094787  1.056120  0.000266
1.005564
22             0.11225  0.00525    0.088235  0.786061 -0.001429
0.973661
23             0.11225  0.00600    0.127660  1.137279  0.000724
1.017665
24             0.11225  0.00775    0.086351  0.769274 -0.002324
0.971653
25             0.11225  0.00550    0.084291  0.750924 -0.001824
0.969468
26             0.15400  0.01350    0.120267  0.780956 -0.003786
0.961656
27             0.11225  0.01350    0.087662  0.780956 -0.003786
0.973050
28             0.08975  0.00875    0.077951  0.868535 -0.001324
0.987204
29             0.11225  0.00875    0.097493  0.868535 -0.001324
0.983649
30             0.15400  0.00900    0.151899  0.986355 -0.000124
0.997522
31             0.08975  0.00525    0.088235  0.983123 -0.000090
0.998339
32             0.15400  0.00925    0.155462  1.009495  0.000087
1.001731
33             0.08975  0.00650    0.109244  1.217200  0.001160
1.021884
34             0.15400  0.00625    0.132979  0.863498 -0.000988
0.975755
35             0.08975  0.00500    0.076628  0.853798 -0.000856
0.985789
36             0.15400  0.01050    0.116992  0.759686 -0.003321
0.958088
```

```
37             0.15400  0.00750    0.114943  0.746380 -0.002549
0.955870
38             0.08975  0.00525    0.080460  0.896488 -0.000606
0.989897
39             0.15400  0.00900    0.100279  0.651159 -0.004821
0.940291

    zhangs_metric
0        0.044528
1        0.112589
2       -0.193904
3       -0.109260
4       -0.096373
5       -0.264551
6       -0.326170
7        0.085472
8        0.207755
9       -0.163849
10      -0.307534
11      -0.308920
12      -0.115955
13      -0.262396
14      -0.266168
15      -0.215213
16      -0.208259
17      -0.212717
18      -0.393583
19      -0.030472
20      -0.208659
21       0.056097
22      -0.224437
23       0.126661
24      -0.247838
25      -0.261909
26      -0.240091
27      -0.248989
28      -0.145667
29      -0.142579
30      -0.014491
31      -0.017926
32       0.010000
33       0.189731
34      -0.142276
35      -0.154828
36      -0.257898
37      -0.266604
38      -0.109944
39      -0.370493
```

```
Association Rules for Subset 2:
              antecedents          consequents  antecedent support  \
0          (bottled beer)         (whole milk)             0.04075   
1         (bottled water)  (other vegetables)             0.05775   
2         (bottled water)         (whole milk)             0.05775   
3           (brown bread)  (other vegetables)             0.03900   
4           (brown bread)         (whole milk)             0.03900   
5                (butter)         (whole milk)             0.03375   
6           (canned beer)         (whole milk)             0.04375   
7          (citrus fruit)         (rolls/buns)             0.05350   
8          (citrus fruit)         (whole milk)             0.05350   
9          (citrus fruit)             (yogurt)             0.05350   
10          (frankfurter)  (other vegetables)             0.04250   
11          (frankfurter)         (rolls/buns)             0.04250   
12          (frankfurter)         (whole milk)             0.04250   
13            (pip fruit)  (other vegetables)             0.04875   
14           (rolls/buns)  (other vegetables)             0.11150   
15     (other vegetables)         (rolls/buns)             0.12750   
16      (root vegetables)  (other vegetables)             0.07200   
17              (sausage)  (other vegetables)             0.06125   
18        (shopping bags)  (other vegetables)             0.05250   
19     (other vegetables)               (soda)             0.12750   
20                 (soda)  (other vegetables)             0.09750   
21       (tropical fruit)  (other vegetables)             0.07200   
22   (whipped/sour cream)  (other vegetables)             0.04925   
23           (whole milk)  (other vegetables)             0.15800   
24     (other vegetables)         (whole milk)             0.12750   
25               (yogurt)  (other vegetables)             0.08625   
26               (pastry)         (whole milk)             0.04900   
27            (pip fruit)         (rolls/buns)             0.04875   
28            (pip fruit)         (whole milk)             0.04875   
29      (root vegetables)         (rolls/buns)             0.07200   
30              (sausage)         (rolls/buns)             0.06125   
31        (shopping bags)         (rolls/buns)             0.05250   
32           (rolls/buns)               (soda)             0.11150   
33                 (soda)         (rolls/buns)             0.09750   
34       (tropical fruit)         (rolls/buns)             0.07200   
35           (rolls/buns)         (whole milk)             0.11150   
36           (whole milk)         (rolls/buns)             0.15800   
37           (rolls/buns)             (yogurt)             0.11150   
38               (yogurt)         (rolls/buns)             0.08625   
39      (root vegetables)         (whole milk)             0.07200   
40              (sausage)               (soda)             0.06125   
41              (sausage)         (whole milk)             0.06125   
42              (sausage)             (yogurt)             0.06125   
43        (shopping bags)               (soda)             0.05250   
44        (shopping bags)         (whole milk)             0.05250   
45       (tropical fruit)               (soda)             0.07200   
46           (whole milk)               (soda)             0.15800   
```

```
47                 (soda)        (whole milk)                0.09750
48                 (soda)            (yogurt)                0.09750
49               (yogurt)              (soda)                0.08625
50       (tropical fruit)        (whole milk)                0.07200
51       (tropical fruit)            (yogurt)                0.07200
52               (yogurt)    (tropical fruit)                0.08625
53               (yogurt)        (whole milk)                0.08625

    consequent support  support  confidence      lift  leverage
conviction  \
0              0.15800  0.00550    0.134969  0.854236 -0.000939
0.973376
1              0.12750  0.00525    0.090909  0.713012 -0.002113
0.959750
2              0.15800  0.00675    0.116883  0.739767 -0.002375
0.953441
3              0.12750  0.00500    0.128205  1.005530  0.000027
1.000809
4              0.15800  0.00500    0.128205  0.811425 -0.001162
0.965824
5              0.15800  0.00550    0.162963  1.031411  0.000167
1.005929
6              0.15800  0.00575    0.131429  0.831826 -0.001162
0.969408
7              0.11150  0.00550    0.102804  0.922007 -0.000465
0.990307
8              0.15800  0.00875    0.163551  1.035135  0.000297
1.006637
9              0.08625  0.00625    0.116822  1.354463  0.001636
1.034616
10             0.12750  0.00550    0.129412  1.014994  0.000081
1.002196
11             0.11150  0.00500    0.117647  1.055131  0.000261
1.006967
12             0.15800  0.00700    0.164706  1.042442  0.000285
1.008028
13             0.12750  0.00800    0.164103  1.287079  0.001784
1.043788
14             0.12750  0.01300    0.116592  0.914446 -0.001216
0.987652
15             0.11150  0.01300    0.101961  0.914446 -0.001216
0.989378
16             0.12750  0.00550    0.076389  0.599129 -0.003680
0.944662
17             0.12750  0.00625    0.102041  0.800320 -0.001559
0.971648
18             0.12750  0.00700    0.133333  1.045752  0.000306
1.006731
19             0.09750  0.01025    0.080392  0.824535 -0.002181
```

```
0.981397
20              0.12750  0.01025    0.105128  0.824535 -0.002181
0.975000
21              0.12750  0.00750    0.104167  0.816993 -0.001680
0.973953
22              0.12750  0.00625    0.126904  0.995322 -0.000029
0.999317
23              0.12750  0.01750    0.110759  0.868702 -0.002645
0.981174
24              0.15800  0.01750    0.137255  0.868702 -0.002645
0.975955
25              0.12750  0.00950    0.110145  0.863882 -0.001497
0.980497
26              0.15800  0.00625    0.127551  0.807285 -0.001492
0.965099
27              0.11150  0.00550    0.112821  1.011843  0.000064
1.001488
28              0.15800  0.00700    0.143590  0.908796 -0.000702
0.983174
29              0.11150  0.00725    0.100694  0.903089 -0.000778
0.987985
30              0.11150  0.00550    0.089796  0.805345 -0.001329
0.976155
31              0.11150  0.00525    0.100000  0.896861 -0.000604
0.987222
32              0.09750  0.00875    0.078475  0.804875 -0.002121
0.979355
33              0.11150  0.00875    0.089744  0.804875 -0.002121
0.976099
34              0.11150  0.00650    0.090278  0.809666 -0.001528
0.976672
35              0.15800  0.01350    0.121076  0.766305 -0.004117
0.957990
36              0.11150  0.01350    0.085443  0.766305 -0.004117
0.971509
37              0.08625  0.00900    0.080717  0.935855 -0.000617
0.993982
38              0.11150  0.00900    0.104348  0.935855 -0.000617
0.992015
39              0.15800  0.00750    0.104167  0.659283 -0.003876
0.939907
40              0.09750  0.00525    0.085714  0.879121 -0.000722
0.987109
41              0.15800  0.00850    0.138776  0.878326 -0.001177
0.977678
42              0.08625  0.00525    0.085714  0.993789 -0.000033
0.999414
43              0.09750  0.00525    0.100000  1.025641  0.000131
1.002778
```

```
44              0.15800  0.00875    0.166667  1.054852  0.000455
1.010400
45              0.09750  0.00700    0.097222  0.997151 -0.000020
0.999692
46              0.09750  0.01225    0.077532  0.795196 -0.003155
0.978353
47              0.15800  0.01225    0.125641  0.795196 -0.003155
0.962991
48              0.08625  0.00850    0.087179  1.010777  0.000091
1.001018
49              0.09750  0.00850    0.098551  1.010777  0.000091
1.001166
50              0.15800  0.00875    0.121528  0.769163 -0.002626
0.958482
51              0.08625  0.00675    0.093750  1.086957  0.000540
1.008276
52              0.07200  0.00675    0.078261  1.086957  0.000540
1.006792
53              0.15800  0.01075    0.124638  0.788846 -0.002877
0.961887

    zhangs_metric
0       -0.151021
1       -0.299312
2       -0.271847
3        0.005723
4       -0.194738
5        0.031518
6       -0.174525
7       -0.082040
8        0.035861
9        0.276492
10       0.015428
11       0.054569
12       0.042521
13       0.234478
14      -0.095267
15      -0.096845
16      -0.418944
17      -0.209973
18       0.046174
19      -0.196078
20      -0.190804
21      -0.194444
22      -0.004919
23      -0.152186
24      -0.147652
25      -0.147077
26      -0.200652
```

```
27        0.012304
28       -0.095432
29       -0.103650
30       -0.204755
31       -0.108235
32       -0.214362
33       -0.211741
34       -0.202116
35       -0.255528
36       -0.265887
37       -0.071618
38       -0.069777
39       -0.357697
40       -0.127758
41       -0.128592
42       -0.006614
43        0.026385
44        0.054881
45       -0.003069
46       -0.234233
47       -0.222017
48        0.011814
49        0.011668
50       -0.244370
51        0.086207
52        0.087551
53       -0.226569

Common Association Rules:
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

1. Image Classification using CNN

Construction of CNN and plotting of the graph

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Define the model
model = keras.Sequential([
    layers.Conv2D(8, (3, 3), activation='relu', input_shape=(28, 28,
1)),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(16, activation='relu'),
    layers.Dense(4, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Display the model summary
model.summary()
```

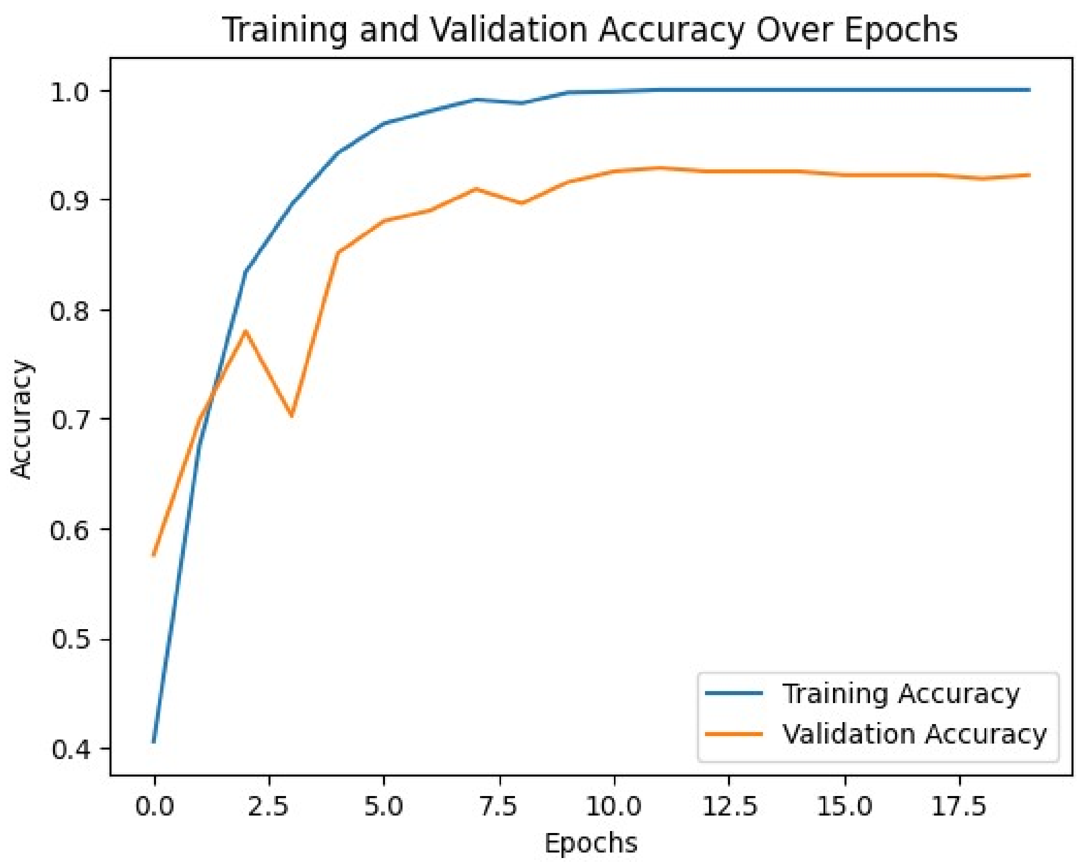
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
```

```
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 8)         80

 max_pooling2d (MaxPooling2  (None, 13, 13, 8)         0
 D)

 flatten (Flatten)           (None, 1352)              0

 dense (Dense)               (None, 16)                21648

 dense_1 (Dense)             (None, 4)                 68

=================================================================
Total params: 21796 (85.14 KB)
Trainable params: 21796 (85.14 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________

import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import categorical_crossentropy
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical


dataset_path = r'/content/drive/MyDrive/ResizedImages'

def load_images_and_labels(directory, target_size=(100, 100)):
    image_data = []
    label_data = []
    label_mapping = {
        'n02089078-black-and-tan_coonhound': 0,
        'n02091831-Saluki': 1,
        'n02092002-Scottish_deerhound': 2,
        'n02095314-wire-haired_fox_terrier': 3
    }
```

```
    for folder_name in os.listdir(directory):
        folder_path = os.path.join(directory, folder_name)
        if os.path.isdir(folder_path) and folder_name in
label_mapping:
            label = folder_name
            encoded_label = label_mapping[label]
            for filename in os.listdir(folder_path):
                img_path = os.path.join(folder_path, filename)
                img = cv2.imread(img_path)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                img = cv2.resize(img, target_size)  # Resize image to
target size
                img = img / 255.0
                image_data.append(img)
                label_data.append(encoded_label)

    return np.array(image_data), np.array(label_data)


images_data, labels_data = load_images_and_labels(dataset_path)


categorical_labels = to_categorical(labels_data, num_classes=4)


X_train_data, X_val_data, y_train_labels, y_val_labels =
train_test_split(images_data, categorical_labels, test_size=0.2,
random_state=42)

model_classifier = Sequential()
model_classifier.add(Conv2D(8, (3, 3), activation='relu',
input_shape=(100, 100, 3)))
model_classifier.add(MaxPooling2D(pool_size=(2, 2)))
model_classifier.add(Flatten())
model_classifier.add(Dense(16, activation='relu'))
model_classifier.add(Dense(4, activation='softmax'))


model_classifier.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
model_classifier.summary()


batch_size_train = 32
epochs_train = 20
history_train = model_classifier.fit(X_train_data, y_train_labels,
batch_size=batch_size_train, epochs=epochs_train,
validation_data=(X_val_data, y_val_labels))


plt.plot(history_train.history['accuracy'], label='Training Accuracy')
```

```
plt.plot(history_train.history['val_accuracy'], label='Validation
Accuracy')
plt.title('Training and Validation Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_2 (Conv2D)           (None, 98, 98, 8)         224       

 max_pooling2d_2 (MaxPoolin  (None, 49, 49, 8)         0         
 g2D)                                                            

 flatten_2 (Flatten)         (None, 19208)             0         

 dense_4 (Dense)             (None, 16)                307344    

 dense_5 (Dense)             (None, 4)                 68        

=================================================================
Total params: 307636 (1.17 MB)
Trainable params: 307636 (1.17 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/20
39/39 [==============================] - 5s 114ms/step - loss: 1.2666
- accuracy: 0.4055 - val_loss: 1.0873 - val_accuracy: 0.5761
Epoch 2/20
39/39 [==============================] - 4s 112ms/step - loss: 0.8395
- accuracy: 0.6764 - val_loss: 0.7842 - val_accuracy: 0.6990
Epoch 3/20
39/39 [==============================] - 6s 158ms/step - loss: 0.5343
- accuracy: 0.8337 - val_loss: 0.6108 - val_accuracy: 0.7799
Epoch 4/20
39/39 [==============================] - 4s 111ms/step - loss: 0.3773
- accuracy: 0.8954 - val_loss: 0.7038 - val_accuracy: 0.7023
Epoch 5/20
39/39 [==============================] - 5s 118ms/step - loss: 0.2701
- accuracy: 0.9424 - val_loss: 0.4968 - val_accuracy: 0.8511
```

```
Epoch 6/20
39/39 [==============================] - 5s 140ms/step - loss: 0.2007
- accuracy: 0.9692 - val_loss: 0.4270 - val_accuracy: 0.8803
Epoch 7/20
39/39 [==============================] - 4s 101ms/step - loss: 0.1512
- accuracy: 0.9805 - val_loss: 0.3836 - val_accuracy: 0.8900
Epoch 8/20
39/39 [==============================] - 4s 110ms/step - loss: 0.1138
- accuracy: 0.9911 - val_loss: 0.3514 - val_accuracy: 0.9094
Epoch 9/20
39/39 [==============================] - 6s 149ms/step - loss: 0.1032
- accuracy: 0.9878 - val_loss: 0.3381 - val_accuracy: 0.8964
Epoch 10/20
39/39 [==============================] - 4s 102ms/step - loss: 0.0677
- accuracy: 0.9976 - val_loss: 0.3060 - val_accuracy: 0.9159
Epoch 11/20
39/39 [==============================] - 4s 108ms/step - loss: 0.0470
- accuracy: 0.9984 - val_loss: 0.3006 - val_accuracy: 0.9256
Epoch 12/20
39/39 [==============================] - 6s 155ms/step - loss: 0.0381
- accuracy: 1.0000 - val_loss: 0.2919 - val_accuracy: 0.9288
Epoch 13/20
39/39 [==============================] - 4s 100ms/step - loss: 0.0314
- accuracy: 1.0000 - val_loss: 0.2861 - val_accuracy: 0.9256
Epoch 14/20
39/39 [==============================] - 4s 110ms/step - loss: 0.0245
- accuracy: 1.0000 - val_loss: 0.2883 - val_accuracy: 0.9256
Epoch 15/20
39/39 [==============================] - 6s 150ms/step - loss: 0.0214
- accuracy: 1.0000 - val_loss: 0.2951 - val_accuracy: 0.9256
Epoch 16/20
39/39 [==============================] - 4s 108ms/step - loss: 0.0172
- accuracy: 1.0000 - val_loss: 0.2918 - val_accuracy: 0.9223
Epoch 17/20
39/39 [==============================] - 4s 108ms/step - loss: 0.0148
- accuracy: 1.0000 - val_loss: 0.2951 - val_accuracy: 0.9223
Epoch 18/20
39/39 [==============================] - 6s 167ms/step - loss: 0.0131
- accuracy: 1.0000 - val_loss: 0.2968 - val_accuracy: 0.9223
Epoch 19/20
39/39 [==============================] - 4s 110ms/step - loss: 0.0116
- accuracy: 1.0000 - val_loss: 0.3038 - val_accuracy: 0.9191
Epoch 20/20
39/39 [==============================] - 4s 105ms/step - loss: 0.0106
- accuracy: 1.0000 - val_loss: 0.3103 - val_accuracy: 0.9223
```



Based on Rowan Banner ID - Last digit is 6 - Experiment (b)

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import categorical_crossentropy
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

# Define the dataset path
dataset_path = r'/content/drive/MyDrive/ResizedImages'

def load_images_and_labels(directory, target_size=(100, 100)):
    image_data = []
    label_data = []
    label_mapping = {
        'n02089078-black-and-tan_coonhound': 0,
        'n02091831-Saluki': 1,
```

```
        'n02092002-Scottish_deerhound': 2,
        'n02095314-wire-haired_fox_terrier': 3
    }

    for folder_name in os.listdir(directory):
        folder_path = os.path.join(directory, folder_name)
        if os.path.isdir(folder_path) and folder_name in
label_mapping:
            label = folder_name
            encoded_label = label_mapping[label]
            for filename in os.listdir(folder_path):
                img_path = os.path.join(folder_path, filename)
                img = cv2.imread(img_path)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                img = cv2.resize(img, target_size)  # Resize image to
target size
                img = img / 255.0
                image_data.append(img)
                label_data.append(encoded_label)

    return np.array(image_data), np.array(label_data)


images_data, labels_data = load_images_and_labels(dataset_path)


categorical_labels = to_categorical(labels_data, num_classes=4)


X_train_data, X_val_data, y_train_labels, y_val_labels =
train_test_split(images_data, categorical_labels, test_size=0.2,
random_state=42)

# Define the model with 4 filters
model_4filters = Sequential()
model_4filters.add(Conv2D(4, (3, 3), activation='relu',
input_shape=(100, 100, 3)))
model_4filters.add(MaxPooling2D(pool_size=(2, 2)))
model_4filters.add(Flatten())
model_4filters.add(Dense(16, activation='relu'))
model_4filters.add(Dense(4, activation='softmax'))


model_4filters.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
model_4filters.summary()


batch_size_train = 32
epochs_train = 20
```

```python
history_4filters = model_4filters.fit(X_train_data, y_train_labels,
batch_size=batch_size_train, epochs=epochs_train,
validation_data=(X_val_data, y_val_labels))


# Define the model with 16 filters
model_16filters = Sequential()
model_16filters.add(Conv2D(16, (3, 3), activation='relu',
input_shape=(100, 100, 3)))
model_16filters.add(MaxPooling2D(pool_size=(2, 2)))
model_16filters.add(Flatten())
model_16filters.add(Dense(16, activation='relu'))
model_16filters.add(Dense(4, activation='softmax'))

model_16filters.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
model_16filters.summary()


history_16filters = model_16filters.fit(X_train_data, y_train_labels,
batch_size=batch_size_train, epochs=epochs_train,
validation_data=(X_val_data, y_val_labels))


# Plot the learning curves for the model with 4 filters
plt.plot(history_4filters.history['accuracy'], label='4 Filters
Training Accuracy')
plt.plot(history_4filters.history['val_accuracy'], label='4 Filters
Validation Accuracy')

# Plot the learning curves for the model with 16 filters
plt.plot(history_16filters.history['accuracy'], label='16 Filters
Training Accuracy')
plt.plot(history_16filters.history['val_accuracy'], label='16 Filters
Validation Accuracy')

plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy with Different Numbers of
Filters')
plt.legend()
plt.show()
```

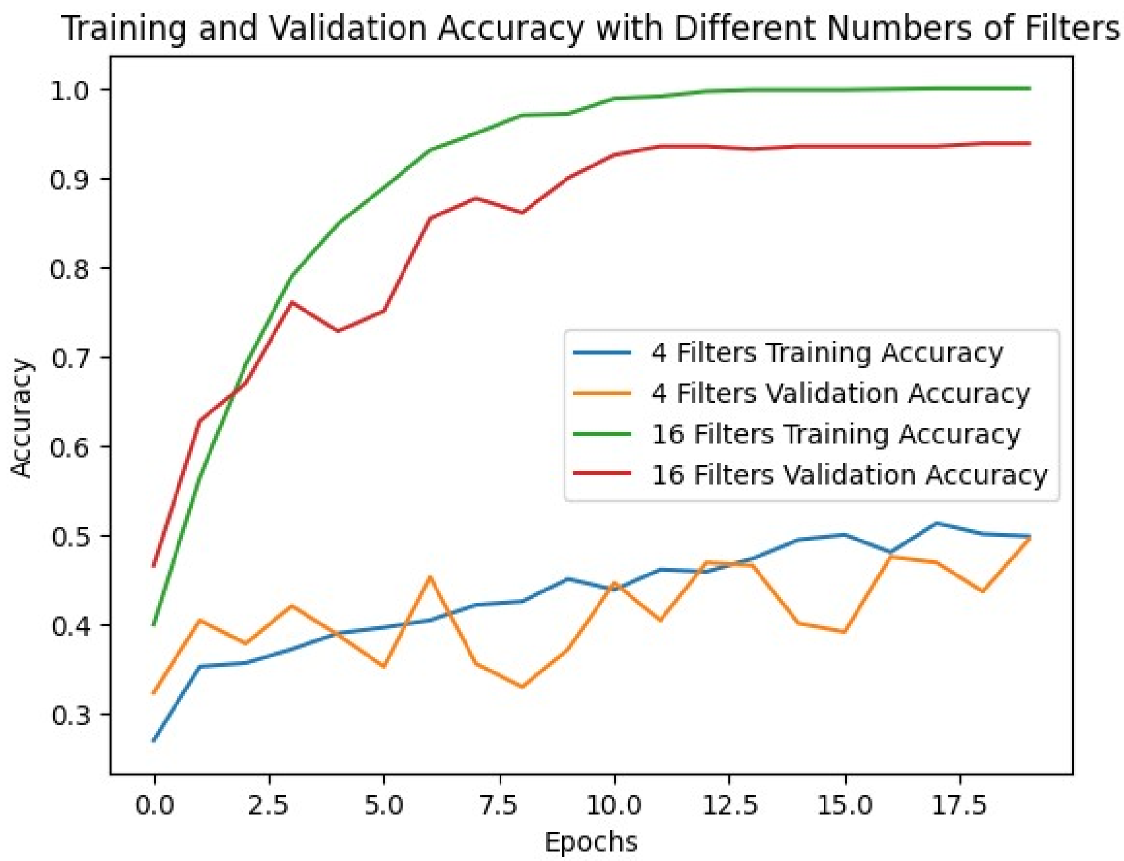
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 98, 98, 4)         112

 max_pooling2d_5 (MaxPoolin  (None, 49, 49, 4)         0
 g2D)

 flatten_5 (Flatten)         (None, 9604)              0

 dense_10 (Dense)            (None, 16)                153680

 dense_11 (Dense)            (None, 4)                 68

=================================================================
Total params: 153860 (601.02 KB)
Trainable params: 153860 (601.02 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/20
39/39 [==============================] - 6s 114ms/step - loss: 1.4746
- accuracy: 0.2701 - val_loss: 1.3827 - val_accuracy: 0.3236
Epoch 2/20
39/39 [==============================] - 4s 107ms/step - loss: 1.3342
- accuracy: 0.3528 - val_loss: 1.3102 - val_accuracy: 0.4045
Epoch 3/20
39/39 [==============================] - 5s 142ms/step - loss: 1.2932
- accuracy: 0.3569 - val_loss: 1.2823 - val_accuracy: 0.3786
Epoch 4/20
39/39 [==============================] - 4s 103ms/step - loss: 1.2534
- accuracy: 0.3723 - val_loss: 1.2622 - val_accuracy: 0.4207
Epoch 5/20
39/39 [==============================] - 4s 107ms/step - loss: 1.2257
- accuracy: 0.3901 - val_loss: 1.2377 - val_accuracy: 0.3883
Epoch 6/20
39/39 [==============================] - 6s 147ms/step - loss: 1.1984
- accuracy: 0.3966 - val_loss: 1.2203 - val_accuracy: 0.3528
Epoch 7/20
39/39 [==============================] - 4s 103ms/step - loss: 1.1758
- accuracy: 0.4047 - val_loss: 1.2261 - val_accuracy: 0.4531
Epoch 8/20
39/39 [==============================] - 4s 105ms/step - loss: 1.1661
- accuracy: 0.4217 - val_loss: 1.1927 - val_accuracy: 0.3560
Epoch 9/20
39/39 [==============================] - 5s 128ms/step - loss: 1.1419
- accuracy: 0.4258 - val_loss: 1.2019 - val_accuracy: 0.3301
Epoch 10/20
39/39 [==============================] - 5s 115ms/step - loss: 1.1455
- accuracy: 0.4509 - val_loss: 1.1722 - val_accuracy: 0.3722
```

```
Epoch 11/20
39/39 [==============================] - 4s 104ms/step - loss: 1.1129
- accuracy: 0.4388 - val_loss: 1.1474 - val_accuracy: 0.4466
Epoch 12/20
39/39 [==============================] - 5s 131ms/step - loss: 1.1033
- accuracy: 0.4615 - val_loss: 1.1464 - val_accuracy: 0.4045
Epoch 13/20
39/39 [==============================] - 5s 115ms/step - loss: 1.0924
- accuracy: 0.4590 - val_loss: 1.1315 - val_accuracy: 0.4693
Epoch 14/20
39/39 [==============================] - 4s 103ms/step - loss: 1.0713
- accuracy: 0.4736 - val_loss: 1.1188 - val_accuracy: 0.4660
Epoch 15/20
39/39 [==============================] - 5s 119ms/step - loss: 1.0532
- accuracy: 0.4947 - val_loss: 1.1160 - val_accuracy: 0.4013
Epoch 16/20
39/39 [==============================] - 5s 124ms/step - loss: 1.0698
- accuracy: 0.5004 - val_loss: 1.1111 - val_accuracy: 0.3916
Epoch 17/20
39/39 [==============================] - 4s 98ms/step - loss: 1.0489 -
accuracy: 0.4809 - val_loss: 1.1127 - val_accuracy: 0.4757
Epoch 18/20
39/39 [==============================] - 4s 103ms/step - loss: 1.0283
- accuracy: 0.5134 - val_loss: 1.0846 - val_accuracy: 0.4693
Epoch 19/20
39/39 [==============================] - 5s 137ms/step - loss: 1.0251
- accuracy: 0.5012 - val_loss: 1.0782 - val_accuracy: 0.4369
Epoch 20/20
39/39 [==============================] - 4s 103ms/step - loss: 0.9981
- accuracy: 0.4988 - val_loss: 1.0730 - val_accuracy: 0.4951
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_6 (Conv2D)           (None, 98, 98, 16)        448       

 max_pooling2d_6 (MaxPoolin  (None, 49, 49, 16)        0         
 g2D)                                                            

 flatten_6 (Flatten)         (None, 38416)             0         

 dense_12 (Dense)            (None, 16)                614672    

 dense_13 (Dense)            (None, 4)                 68        

=================================================================
Total params: 615188 (2.35 MB)
Trainable params: 615188 (2.35 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
Epoch 1/20
39/39 [==============================] - 8s 177ms/step - loss: 1.2996
- accuracy: 0.3998 - val_loss: 1.1335 - val_accuracy: 0.4660
Epoch 2/20
39/39 [==============================] - 5s 129ms/step - loss: 0.9831
- accuracy: 0.5645 - val_loss: 0.8816 - val_accuracy: 0.6278
Epoch 3/20
39/39 [==============================] - 6s 168ms/step - loss: 0.7525
- accuracy: 0.6910 - val_loss: 0.8179 - val_accuracy: 0.6699
Epoch 4/20
39/39 [==============================] - 5s 128ms/step - loss: 0.5722
- accuracy: 0.7899 - val_loss: 0.6465 - val_accuracy: 0.7605
Epoch 5/20
39/39 [==============================] - 5s 121ms/step - loss: 0.4617
- accuracy: 0.8483 - val_loss: 0.6846 - val_accuracy: 0.7282
Epoch 6/20
39/39 [==============================] - 7s 174ms/step - loss: 0.3569
- accuracy: 0.8889 - val_loss: 0.6555 - val_accuracy: 0.7508
Epoch 7/20
39/39 [==============================] - 5s 129ms/step - loss: 0.2756
- accuracy: 0.9311 - val_loss: 0.4285 - val_accuracy: 0.8544
Epoch 8/20
39/39 [==============================] - 6s 167ms/step - loss: 0.2136
- accuracy: 0.9497 - val_loss: 0.3907 - val_accuracy: 0.8770
Epoch 9/20
39/39 [==============================] - 5s 131ms/step - loss: 0.1613
- accuracy: 0.9700 - val_loss: 0.3862 - val_accuracy: 0.8608
Epoch 10/20
39/39 [==============================] - 5s 129ms/step - loss: 0.1355
- accuracy: 0.9716 - val_loss: 0.3187 - val_accuracy: 0.8997
Epoch 11/20
39/39 [==============================] - 6s 168ms/step - loss: 0.1012
- accuracy: 0.9886 - val_loss: 0.2987 - val_accuracy: 0.9256
Epoch 12/20
39/39 [==============================] - 5s 132ms/step - loss: 0.0866
- accuracy: 0.9911 - val_loss: 0.2747 - val_accuracy: 0.9353
Epoch 13/20
39/39 [==============================] - 7s 182ms/step - loss: 0.0554
- accuracy: 0.9968 - val_loss: 0.2792 - val_accuracy: 0.9353
Epoch 14/20
39/39 [==============================] - 6s 144ms/step - loss: 0.0468
- accuracy: 0.9984 - val_loss: 0.2554 - val_accuracy: 0.9320
Epoch 15/20
39/39 [==============================] - 6s 166ms/step - loss: 0.0339
- accuracy: 0.9984 - val_loss: 0.2929 - val_accuracy: 0.9353
Epoch 16/20
39/39 [==============================] - 6s 158ms/step - loss: 0.0336
- accuracy: 0.9984 - val_loss: 0.2443 - val_accuracy: 0.9353
Epoch 17/20
39/39 [==============================] - 5s 140ms/step - loss: 0.0221
```

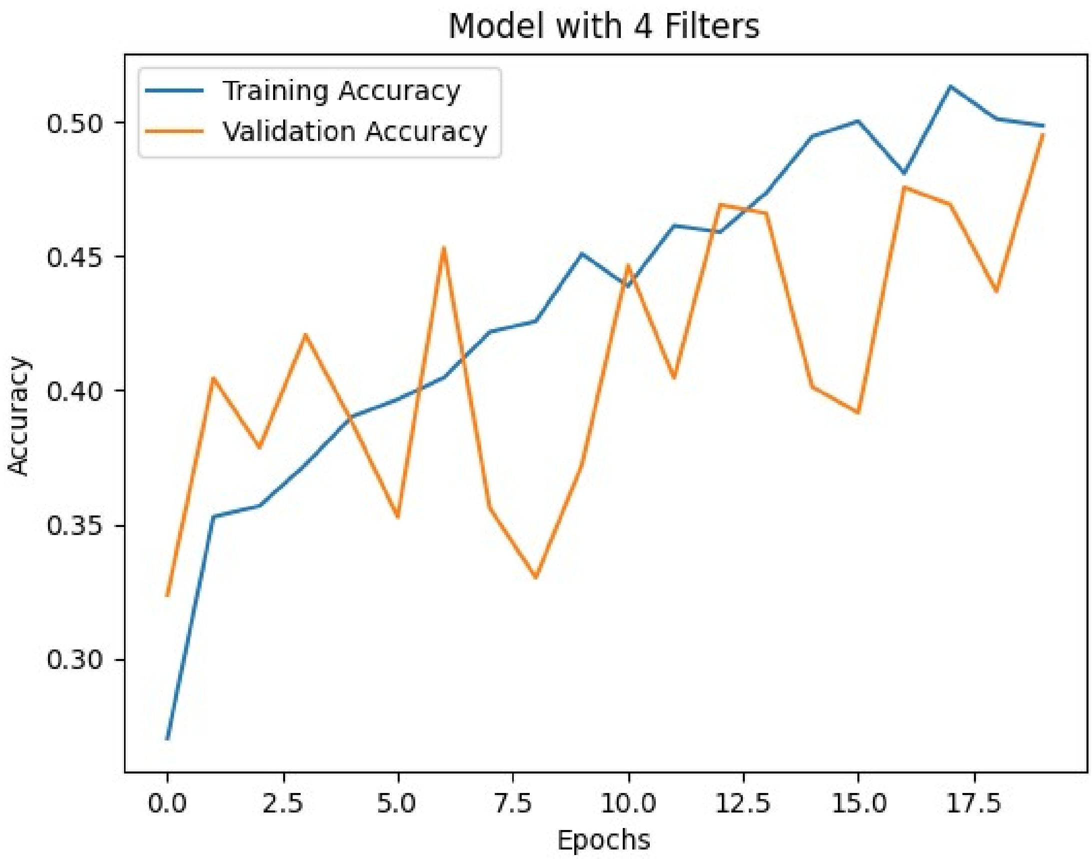
```
- accuracy: 0.9992 - val_loss: 0.2397 - val_accuracy: 0.9353
Epoch 18/20
39/39 [==============================] - 7s 176ms/step - loss: 0.0201
- accuracy: 1.0000 - val_loss: 0.2497 - val_accuracy: 0.9353
Epoch 19/20
39/39 [==============================] - 5s 124ms/step - loss: 0.0165
- accuracy: 1.0000 - val_loss: 0.2438 - val_accuracy: 0.9385
Epoch 20/20
39/39 [==============================] - 7s 171ms/step - loss: 0.0132
- accuracy: 1.0000 - val_loss: 0.2452 - val_accuracy: 0.9385
```

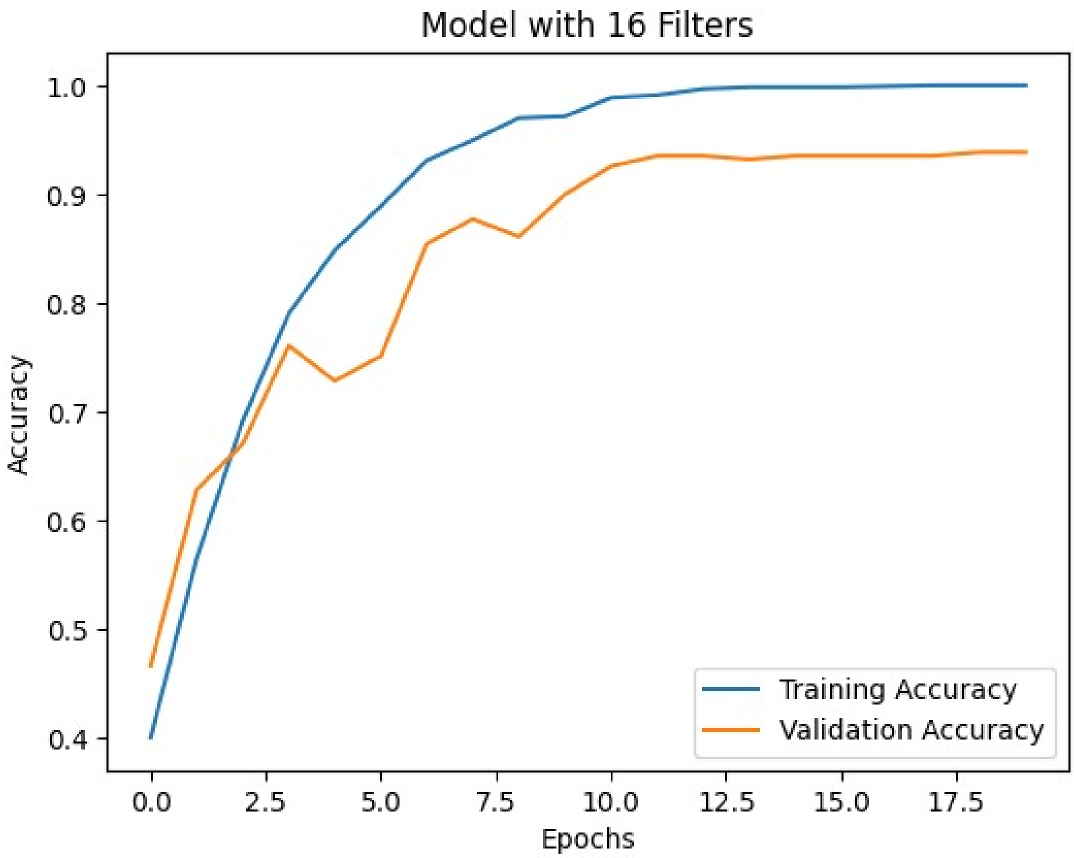



```
import matplotlib.pyplot as plt

# Function to plot learning curves
def plot_learning_curves(history, title):
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
```

```
    plt.show()

# Plot learning curves for model with 4 filters
plot_learning_curves(history_4filters, "Model with 4 Filters")

# Plot learning curves for model with 16 filters
plot_learning_curves(history_16filters, "Model with 16 Filters")
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```





Model with 16 Filters
Accuracy
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Training Accuracy
Validation Accuracy